Prashanta Foundation
Village Baprola, Najafgarh, New Delhi-110043

Ph. 9990238023
9911238023

By :- Prashant Solanki

Chapter 1

Overview Of C

INTRODUCTION

'C' was invented and first implemented by Dennis M. Ritchie in 1970. It is called 'C' because their was another language present called 'Basic Combined Programming Language' (BCPL), which is also called 'B' developed by martin Richards.

'C' is a high level language but it is also called a middle level language because it contains the best elements of high level language with the control and flexibility of Assembly language, which is a low level language. 'C' is programmer's language because it gives the programer what the programer wants.

'C' language is well suited for structured programming, thus requiring the user of think of a problem in terms of function modules or blocks. A proper collection of these modules would make a complete program. Another important feature of 'C' is its ability to extend itself. A 'C' program is basically a collection of functions that are supported by the 'C' library. We can continuously add our own functions to the 'C' library. With the availability of a large number of functions, the programming task becomes simple.

OPERATORS AND EXPRESSIONS

An operator is a symbol or letter used to indicate a specific operation on variables in a program. For example symbols +, - , / , * etc., are the operators and used to calculate two ***operands.***

An expression is a combination of operands connected by operators and parenthesis. For example in **A + B**, A and B are operands and **'+'** is an operator

'C' supports a rich set of operators. 'C' operators can be classified into a number of categories:

a) Arithmetic Operators
b) Relational Operators
c) Logical Operators
d) Conditional Operators
e) Assignment Operator
f) Special Operators

a) Arithmetic Operators

Operator	*Meaning*
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo division (find reminder)

b) Relational Operators

Operator	*Meaning*
<	Less than
<=	Less than equal
>	Greater than

>=	Greater than equal
= =	Equal
!=	not equal

c) Logical Operators

Operator	*Meaning*
&&	AND
||	OR
!	NOT

d) Conditional Operator

'C' provides a conditional operator **(? :)** which can help the programmer in performing simple conditional operations. The general form is :-

exp 1 ? exp 2 : exp 3

for example :- z = (x > y) ? x : y;

In the above example the expression x>y is evaluated. If x is greater than y, then x is assigned into z otherwise y.

e) Assignment Operator

The assignment operator (=) is used to assign the result of an expression or a value to a variable. During assignment operation, 'C' converts the type of value on the right-hand side to the type on the left. For example

a = 6; or a = a+1;

f) Special Operators

(i) Increment/Decrement Operator :- There are two special and very useful operators the 'C' has, which are not generally found in any other languages. These operators are ***increment*** and ***decrement*** operators (++ and --). The operator '++' adds 1 to the operand while '--' subtracts 1. For example :

a++; or a = a + 1;
a--; or a = a - 1;

(ii) The shorthand assignment Operator :- The shorthand assignment operator is used to write the assignment statement in short way. For example

a += 2; => a = a + 2
a *= 5; => a = a * 5

include

The *'# include*' directive include or attach another file in the program. The functions of the included file can be used in the program. You can include header files with your program in the following manner:-

 # include <stdio.h>

HEADER FILES

Header files contains the predefined functions. If you want to use a function or work on a function so, first you have to include the header file with your program.

main( )

The *main()* is a special function of 'C' which tell the computer from where the program starts. Every program must have exactly one main function. The opening brace '{' of the *main* function

indicate the beginning of the function and closing brace '}' indicates the end of the function. All the statements between these two braces form the *'function body'*.

VARIABLE

A variable is a named location in the memory that is used to hold a value that may be modified by the program. All variables must be declared before they can be used. You can declare a variable like: **<data type> <variable_name list>**

You can declare a variable in the following two manner :-

(i) Local Variable :- Variables declared inside any function is called *local variable*. Local variables are visible and meaningful only inside the functions in which they are declared. They are not known to other functions.

(ii) Global Variable :- Variables declared outside of any function is called *global variable.* The variable can be used inside any function, and it is usually best to declare global variable at the top of the program.

DATA TYPES

Data can be of many types e.g. character, integer, real, string etc., and you have to declare the type of a variable before using it. Following are the size and the range of basic data types :-

Data Type	Size (in byte)	Format Specifier	Range
char (Single Character)	1	%c	-128 to 127
int (Integer)	2	%d	-326768 to 32767
float (Decimal numbers)	4	%f	3.4×10^{-38} to 3.4×10^{38}
char [size] (String)	according to size	%s	

BACKSLASH CHARACTER CONSTANTS

C supports some special backslash character constants that are used in output functions.

Constant	Meaning
'\a'	audible alert (bell)
'\b'	back space
'\f'	form feed
'\n'	new line
'\r'	carriage return
'\t'	horizontal tab
'\v'	vertical tab
'\''	single quote
'\" '	double quote
'\?'	question mark
'\\'	backslash
'\0'	NULL

Chapter 2

C Programming

C is made up entirely of building blocks which have a particular `shape' or form. The form is the same everywhere in a program, whether it is the form of the main program or of a subroutine. A program is made up of functions, functions are made up of statements and declarations surrounded by curly braces { }. The basic building block in a C program is the function. Every C program is a collection of one or more functions, written in some arbitrary order. One and only one of these functions in the program must have the name main().
The function main() does not have to be at the top of a program so a C program does not necessarily start at line 1. It always starts where main() is. Also, the function main() cannot be called from any other function in the program. Only the operating system can call the function main(): this is how a C program is started.

Comments : Comments are a way of inserting remarks and reminders into a program without affecting its content. Comments do not have a fixed place in a program: the compiler treats them as though they were white space or blank characters and they are consequently ignored. Programs can contain any number of comments without losing speed. This is because comments are stripped out of a source program by the compiler when it converts the source program into machine code. Comments are marked out or delimited by the following pairs of characters:

/* ..... comment .....*/ ***or*** // Single line comment

Because a comment is skipped over as though it were a single space, it can be placed anywhere where spaces are valid characters, even in the middle of a statement, though this is not to be encouraged. You should try to minimize the use of comments in a program while trying to maximize the readability of the program. If there are too many comments you obscure your code and it is the code which is the main message in a program.

2.1 Sample Program

Example 1. ***Input any two numbers and find their sum.***

```
#include<stdio.h>
#include<conio.h>
void main( )
{
        clrscr( );              // Clears the screen
        int m, n, s;            // Declaring variables
        printf("Enter the first no :-");      // Output operator
        scanf("%d",&m);                       // Input operator
        printf("Enter the second no :-");
        scanf("%d",&n);
        s = m + n;
        printf("\n The sum of %d and %d is %d",m, n, s);
        getch();                // Waits until any key not pressed
}
```

Chapter 3

Control Statements

DECISION MAKING WITH IF STATEMENT

The **if** statement is a powerful decision making statement and is used to control the flow of execution of statements. An **if** statement tests a particular condition and execute the given statement according to result. The **if** statement may be implemented in different forms depending on the complexity of conditions to be tested. There are four general forms of **if** statements :

i) Simple **if** statement
ii) **if...else** statement
iii) Nested **if...else** statement
iv) **else...if** ladder.

i) Simple *if* Statement :- In this statement, if the given condition become true then, the statement inside the **if** statement body will execute. The syntax is :

```
if (<condition>)
{
        <statement>;
}
```

Example 1. ***WAP to check whether an inputted number positive or negative.***

```
void main( )
{
     clrscr( );
     char n;
     printf("Enter any number :-");
     scanf("%d",&n);
      if (n>=0)
            printf("\n The number is positive");
      if (n<0)
            printf("\n The number is negative");
       getch( );
}
```

ii) *if...else* statement :- This statement will use in the case, when we have two conditions for true and false. If the given condition is true then it will execute only the true statement otherwise the false statement. The syntax is :-

```
if (<condition>)
{
     <true statement>
}
else
{
     <false statement>
}
```

Example 2. ***Input any two nos and find the maximum number.***

```
void main( )
{
   clrscr( );
   int m, n, ma;
   printf("Enter first number :- ");
   scanf("%d",&m);
   printf("Enter second number :-");
   scanf("%d",&n);
   if (m > n)
   {
      ma = m;
   }
   else
   {
      ma = n;
   }
   printf("\n The maximum number is %d",ma);
   getch ( );
}
```

iii) Nested *if...else* Statement :- When an **if...else** statement comes within another, is called nested **if...else** statement. The syntax is :

```
if (<condition>)
{
        <statement>;
        if (<condition>)
        {   <statement>; }
        else
        {   <statement>;  }
}
else
{
        <statement>;
        if (<condition>)
        {    <statement>; }
        else
        {     <statement>;  }
}
```

Example 3. ***Find the largest among three values.***

```
void main( )
{
    clrscr( );
    int a, b, c, m;
    printf("Enter any three numbers :-");
    scanf("%d %d %d",&a,&b,&c);
```

```
        if (a > b)
        {
           if (a > c)
                m = a;
            else
                m = c;
        }
        else
        {
            if(c > b)
                m = c;
             else
                m = b;
         }
        printf("The largest number is %d",m);
         getch( );
}
```

iv) *else....if* ladder :- If you have more than two conditions then the **else...if** ladder will use. You can also specify all the conditions through simple **if** statement. In that case, all the statements will check by the complier and this process consume time. But in other hand, if the first condition is true, all the rest statements will not check. The syntax of this type is :

```
if (<condition 1>)
{
     <statement>;
}
else if (<condition 2>)
{
     <statement>;
}
else if(<condition n>)
{
    <statement>;
}
else
{
    <statement>;
}
```

Example 4. ***WAP to input numbers between 1 to 7 and print it's day name.***

```
void main( )
{
   clrscr( );
   int n;
   printf("Enter any number :-");
   scanf("%d",&n);
```

```
    if (n==1)
      printf("\n\t The day is Sunday");
    else if (n==2)
      printf("\n\t The day is Monday");
    else if (n==3)
      printf("\n\t The day is Tuesday");
    else if (n==4)
      printf("\n\t The day is Wednesday");
    else if (n==5)
      printf("\n\t The day is Thursday");
    else if (n==6)
      printf("\n\t The day is Friday");
    else if (n==7)
      printf("\n\t The day is Saturday");
    else
      printf("\n\t Wrong Choice!!!");
     getch( );
}
```

THE SWITCH STATEMENT

The **switch** statement is used instead of **if...else if...else** statement. The use of multiple **if...else** nested statements makes the program more complex and difficult to understand. The expression in the switch part is evaluated to give desired value of **int** or **char**. **Float** is not allowed in expression. The value in expression of switch, it matches with the expression of case statement and control goes inside that block and executes that block. If no expression matches then control goes on **default** expression. The general form of this statement is :-

```
switch (<expression>)
{
        case <constant 1> :
                                <statement>;
                                break;
        case <constant n> :
                                <statement>;
                                break;
        default :
                                <statement>;
}
```

Example 5. ***WAP to make a calculator which is able to do the operations of +, -, *, / using case control.***

```
void main( )
{
   clrscr( );
  char optr;
  int a, b;
  printf("Enter any one operator (+ - * /):- ");
 scanf("%c",&optr);
```

```
        printf("Enter any two values :- ");
        scanf("%d %d",&a,&b);
        switch (optr)
        {
            case '+' :
                        printf("\nThe result is %d", a + b);
            case '-' :
                        printf("\n The result is %d",a - b);
            case '*' :
                        printf("\n The result is %d",a * b);
            case '/' :
                        printf("\n The result is %f ",a / b);
             default :
                        printf("\n\nWrong Choice!!!");
        }
        getch( );
    }
```

Exercise

1. Swap the values of two variables with the use of third variable.
2. Swap the values of two variables without use of third variable.
3. Input any three numbers and find the maximum number between them.
4. Input any number and check whether it is even or odd.
5. Find that whether the sum of three inputted numbers is even or odd.
6. Find the minimum number between five inputted numbers.
7. Input any number between 1 to 12 and print it's month name.
8. Input any character and print that whether it is a vowel or not.
9. WAP to print the grade of a student according to the following rules after inputting the percentage:

Percentage	*Grade*
80 to 100	Distinction
60 to 79	First Division
50 to 59	Second Division
40 to 49	Third Division
0 to 39	Fail

10. Write the code for the following with using case control.
    1.) Sum of two inputted numbers
    2.) Product of three inputted numbers
    3.) Check whether inputted number even or odd.
    4.) Find maximum between three numbers.

    Enter your choice :-

Chapter 4

Looping

The loops allow a set of instructions to be performed repeatedly until a certain condition is fulfilled. 'C' provides three kinds of loops :

i) The **while** loop
ii) The **for** loop
iii) The **do...while** loop

THE *while* LOOP

In **while** loop, first of all, it tests the condition and if the condition evaluates to true, then the body of the loop is executed. After execution of the body, the test condition is once again evaluated and if it is again true, the body executs once again. This process repeat until the test condition finally becomes false and the control is transferred out of the loop. The basic syntax of the while loop is :-

```
<var declaration>;
while (<condition>)
{
        <statement>;
        <processing of loop>;
}
```

Example 1. ***Print 1 to 10 counting***

```
void main( )
{
        clrscr( );
        int i=1;
        while(i<=10)
        {
                printf("\n %d",i);
                i++;
        }
        getch( );
}
```

Exercise :

1. Print the table of 2
2. Print the table of 18
3. WAP to input any number and print it's table.
4. Input any no and check whether it is even or odd. If the number is even then print table of 2 else print table of 5.
5. Input any three numbers and print the table of maximum number.
6. Input any two numbers and print the table of their multiplication.

THE *for* LOOP

The **for** loop is the easiest to understand. All its loop-control elements are gathered in one place. The syntax of this loop is :-

```
for (initialization ; test-condition ; processing)
{
    <statements>;
}
```

Example 2. ***Find the factorial of an inputted number.***

```
void main( )
{
   clrscr( );
   int n, fa=1, i;
   printf("Enter any number :- ");
   scanf("%d",&n);
   for(i=n; i>=1; i--)
   {
       fa *= i;
   }
   printf("\n\nThe factorial is %d",fa);
   getch();
}
```

Example 3. ***Check whether an inputted number prime or not.***

```
void main( )
{
    clrscr( );
    int n, i, flag=0;
    printf("Enter any number :-");
    scanf("%d",&n);
     for(i=2; i<=n/2; i++)
     {
          if (n%i==0)
          {
             flag=1;
             break;
          }
      }
      if (flag==0)
          printf("\n The number is prime");
       else
          printf("\n The number is not prime");
       getch( );
}
```

NESTED LOOP :- When a loop occurs inside another loop is called nested loop.

Example 4. ***Print 2 to 20 tables.***

```
void main( )
{
        clrscr( );
        for(i=2 ; i<=20; i++)
        {
                for(j=1; j<=10; j++)
                {
                        printf("\n %d",i*j);
                }
                getch( );
                clrscr( );
        }
}
```

Example 5. ***Print in following format after inputting any number.***

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
void main( )
{
     clrscr( );
     int n, i, j;
    printf("Enter any Number :-");
    scanf("%d",&n);
     for (i=1; i<=n; i++)
     {
        for (j=1; j<=i; j++)
        {
                printf(" %d", j);
        }
        printf("\n");
     }
     getch( );
}
```

Exercise

1. Print 20 to 2 tables.
2. Input any number and print number to 20 tables.
3. Input any number and print number to 2 tables.
4. Input any two numbers and print minimum to maximum number's table.
5. Input any number and check is it even or odd. If the number is even then print 2 to 20 tables else print 20 to 2 tables.

6. Print in the following format:-

```
1 2 3 4
1 2 3
1 2
1
```

THE *do....while* LOOP

The **for** and **while** loop makes a test of condition before the loop is executed. If the condition is not true then, the body of the loop may not be executed at all. But, The **do** loop evaluate the body of the loop first and at the end of the loop, the test condition is evaluated by the **while** statement. The syntax of this loop is :-

```
do
{
        <statement>;
}while(<condition>);
```

Example 6. ***Write the code of the following program***

1.) Table of 2
2.) Table of 5
3) Table of 8
4.) Exit
Enter your choice :-

```
void main( )
{
        int ch, i;
        do
        {
                clrscr( );
                printf("\n 1.) Table of 2");
                printf("\n 2.) Table of 5");
                printf("\n 3.) Table of 8");
                printf("\n 4.) Exit");
                printf("\n\nEnter your choice :- ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1 :
                                        for(i=1;i<=10;i++)
                                                printf("\n %d",2*i);
                                        getch( );
                                        break;
                        case 2 :
                                        for(i=1;i<=10;i++)
                                                printf("\n %d",5*i);
                                        getch( );
                                        break;
                        case 3 :
```

```
                                        for(i=1;i<=10;i++)
                                            printf("\n %d",8*i);
                                        getch( );
                                        break;
                                case 4 :    break;
                                default :
                                        printf("\n\nWrong Choice!!!!");
                            }
                        }while(ch !=4);
                }
```

Exercise

1. Write the code for the following program

   1.) Inputted numbers table
   2.) 2 to 20 tables
   3.) 20 to 2 tables
   4.) Check whether inputted number prime or not.
   5.) Exit

   Enter your choice :-

2. Write the program to print the bill for the following program

Items	Rates
1.) Tea	Rs. 5.00/-
2.) Burger	Rs. 8.00/-
3.) Dosa	Rs. 25.00/-
4.) Exit	

   Enter your choice:-

   If the choice if less than 4, then input

   Enter the quantity :-

   and print the bill in the following format

   You have ordered 3 tea
   and your bill is Rs. 15.00/- only

Chapter 5

Arrays

An array is a collection of variables of the same data type that are referred to through a common name. The lowest address is known as the first element and the highest address is known as last element of the array. The individual element within the array can be represented by an integer known as **subscript**. There are two types of array :- (i) Single Dimension Array (ii) Double Dimension Array.

5.1 SINGLE DIMENSION ARRAY

An array whose elements are specified by a single subscript is known as single dimensional array. You can declare it as :-

<data type> <variable name> [<size>];

for example :- int n[10];

Example 1. ***WAP to input any 10 numbers in an array and print them.***

```
void main( )
{
    clrscr( );
    int n[10],i;
     for(i=0; i<=9; i++)
     {
         printf("Enter any number:-");
         scanf("%d",&n[i]);
     }
     printf("\nThe array is :- \n");
      for (i=0;i<=9;i++)
      {
              printf("\n%d",n[i]);
      }
       getch( );
}
```

Exercise

1. WAP to input any 10 numbers in an array and print only even numbers.
2. Input any 10 numbers in an array and print only prime numbers.
3. Input numbers in two 10-elements array and print their sum.
4. WAP to input any numbers in 10-elements array and print table of all the numbers.

Example 2. ***Write a program to input any 10 numbers in an array and sort them.***

```
void main( )
{
    clrscr ( );
```

```
int n[10], i, j;
 for(i=0; i <=9 ;i++)
 {
   printf("Enter any number :-");
   scanf("%d",&n[i]);
 }
 for(i=0; i<=9; i++)
 {
     for(j=i+1; j<=9; j++)
     {
          if (n[i] > n[j])
          {
             n[i]=n[i] +n[j];
             n[j]=n[i] - n[j];
             n[i]=n[i] - n[j];
           }
      }
   }
   printf("\nThe sorted array is :- \n");
    for(i=0; i<=9; i++)
       printf("\n%d",n[i]);
    getch();
}
```

(ii) Double Dimension Array :- A double dimensional array is an array in which each element is itself an array. For example, an array Arr[3][3] is 3 by 3 table with 3 rows and 3 columns containing 3x3 elements.

Example 3. ***WAP to input numbers in 3 x 3 elements array and print them.***

```
void main( )
{ clrscr( );
     int n[3][3], i, j;
     for(i=0; i<=2; i++)
     {
          for(j=0; j<=2; j++)
          {
               printf("Enter any number :-");
               scanf("%d",&n[i][j]);
          }
     }
     printf("\nThe matrix is :-\n");
     for(i=0; i<=2; i++)
     {
     for(j=0; j<=2; j++)
     {
          printf("\t %d",n[i][j]);
     }
```

```
                printf("\n");
                }
                getch( );
        }
```

Exercise

1. WAP to input numbers in 3 x 3 elements array and print only prime numbers otherwise 0.
2. WAP to input numbers in two 3 x 3 elements array and print their sum.
3. WAP to input numbers in 3 x 3 elements array and print the diagonal numbers.
4. WAP to input numbers in 3 x 3 elements array and print the upper portion from the diagonal.
5. WAP to input numbers in 3 x 3 elements array and print the lower part from the diagonal.
6. WAP to print the multiplication of two 3 x 3 elements matrix.

Chapter 6

String Handling

As you studied earlier that an array is a collection of variables of the same data type. As we stores integers in an integer array, similarly a group of characters can be stored in a character array. So, a string is a collection or group of characters. A string constant is a one-dimensional array of characters terminated by a ***NULL ('\0')***. For example :

Char str[]={'B', 'H', 'A', 'R', 'T', 'I', '\0'}

The NULL character indicates the end of the string.

Ex. 1) ***W.A.P. to input any string and print it's length.***

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    clrscr( );
    char str[20];
    int len;
    printf("Enter any string :-");
    gets(str);
    for (len=0; str[len]!=NULL; len++);
    printf("\n\n\n The Inputted String is %s ",str);
    printf("\n\n The Length of the String is %d",len);
    getch( );
}
```

The format specifier of the string is ***'%s'***. In the above example we have used ***gets( )*** function instead of ***scanf( )*** because when you input string through ***scanf( )*** function it terminates the string as you put space; it means ***scanf( )*** function is not capable of receiving multi word string (if you put ***Prashant Solanki***, it will accept ***Prashant*** only). But in other hand the ***gets( )*** function accepts the spaces between the string. You can also make the ***scanf( )*** function capable to accept multi word string by writing it in this manner :

```
char n[20];
printf("Enter your full name :-");
scanf("%[^\n]s",n);
```

TWO-D CHARACTER ARRAY

In the above section, we have learnt to put characters in single dimension character array. Every element of the array contains a single character. But we can also use two dimension character array. The following program will help you to understand this :

```
#include<stdio.h>
#include<conio.h>
void main()
{
   clrscr();
  char n[5][10]={"Shivam",
                 "Bunty",
                 "Mohit",
                 "Neeraj",
                 "Suman"};
   for (int i=0;i<=4;i++)
   {
        printf("\n%s",n[i]);
   }
   getch();
}
```

In the above example, array n[5][10] containing the 5 strings of length 10 per string. So, the first subscript of the array contains the number of names and the second subscript contains the length of the names.

Exercise

1. WAP to input any name and reverse it.
2. WAP to input any two strings and compare them wether they are length wise equal or not ?
3. WAP to input any two strings and check wether they are equal or not ?

Chapter 7

Functions

Every program must have a ***main( )*** function to indicate where the program has to begin its execution. It leads to a number of problems if you code the entire program within the ***main( )*** function. The program may become too large and complex and as a result the task of debugging, testing, and maintaining becomes difficult. To avoid this type of problem, divide a program into functional parts, then each part may be independently coded and later combined into a single unit. These subprograms called ***'functions'***.

A function is a self-contained block of code that performs a particular task. Once a function has been designed and packed, it can be treated as a ***'black box'*** that takes some data from the main program and returns a value. The inner details of operation are invisible to the rest of the program. The length of a source program can be reduced by using functions at appropriate places.

For a C++ function, you must do the following:

- Provide a function prototype.
- Call the function.
- Provide a function definition.

(i) Function Prototype :- The declaration of the function is called prototype of a function. The prototype describes the function interface to the compiler. That is, it tells the compiler what type of return value, if any, the function has, and it tells the compiler the number and type of function arguments. A function's prototype contains the name of the function, a list of variables that must be passed to it, and the type of variable it returns, if any. For example :

<return type> <function name> (<parameter list>);

Note :- The variables come with function's name are called parameters and the variables pass at the time of calling of the functions are called arguments. The parameters are function's variable while arguments are programs variable.

(ii) Calling of the Function :- When the function is invoked at the required place in the program, it is known as calling of the function.

(iii) Function Definition :- A *function definition* is the actual function. The definition contains the code that will be executed. The first line of a function definition, called the *function header,* should be identical to the function prototype, with the exception of the semicolon. A function header shouldn't end with a semicolon. In addition, although the argument variable names were optional in the prototype, they must be included in the function header. Following the header is the function body, containing the statements that the function will perform. The function body should start with an opening bracket and end with a closing bracket. If the function return type is anything other than void, a return statement should be included, returning a value matching the return type.

FUNCTIONS RETURNING A VALUE

Every function returns a value at the end of it. The returning value should match the return type. A function can return only one value.

Example 1: Sum of two numbers throgh function.

```
int sum(int, int);                          // function prototype
void main()
{  clrscr();
   int a, b, s;
   printf("Enter any two numbers :- ");
   scanf("%d %d",&a,&b;
   s = sum(a, b);                           // calling of the function
   printf("\nThe sum is %d",s);
   getch();
}
int sum ( int m, int n)                     // function definition
{
   return m + n;
}
```

Exercise

1. W.A.P. to find the cube of an inputted number through function.
2. W.A.P. to find the maximum between three inputted numbers through function.
3. W.A.P. to find the factorial of an inputted number by using a function.
4. W.A.P. to reverse the digits of a number.

FUNCTION NOT RETURNING A VALUE

If you don't want to return any value from the function then put ***void*** before the function. When we make any function as ***void***, it avoids to return any value. In return type function, we print the result in ***main( )*** because the function returns the result but in other hand, we print the result in the function if it is of not-returning type.

Ex. Print the table of an inputted number through function.

```
void table(int n)
{
    for(int i=1;i<=10;i++)
        printf("\n %d ",n * i);
}
void main( )
{   clrscr();
    int n;
    printf("Enter any number :-");
    scanf("%d",&n);
    table(n);
    getch( );
}
```

Exercise

1. W.A.P. to check whether an inputted number is prime or not.
2. W.A.P. to print whether an inputted number even or odd.

Functions works in two manner :- ***Call-by-value*** and ***Call-by-reference.***

Pass / Call-By-Value :- In ***call-by-value,*** when we pass argument list to function at calling time, the function copies the values of the argument to its parameter's variable. If we make any change in function's variable the original values remains unchanged in the program. It is because, function's variable occupies separate memory address instead of using the address of original variables. This ensures that the original data in the calling function cannot be changed accidentally. All the above functions are the examples of this type.

Pass / Call-by-reference :- In this type of function, instead of coping the value of original variables to function's variable, the functions variables (parameters) refers the address of original variables (arguments). In this type of situation when make any change in functions variable the value of original variable has also changed. It is because, both variables (arguments and parameters) refers the same memory address.

For example :-

```
void swap(int &a, int &b)
{       a = a + b;
        b = a - b;
        a = a - b;
}
void main( )
{       clrscr( );
        int m,n;
        printf("Enter any two numbers :-");
        scanf("%d%d",&m,&n);
        swap(m, n);
        printf("\n\nThe value of m is %d",m);
        printf("\n\nThe value of n is %d",n);
        getch( );
}
```

(i) Function with default argument :- There is a facility with the functions that we can declare a function with default argument values. For example :-

```
int sum(int a=0, int b=0)
    return a+b;
void main( )
{
   printf("The sum is %d",sum(6));
    getch( );
}
```

In the above example, at the calling time of the function, value 6 will transfer to a and 0 will transfer to b.

PASSING ARRAY TO FUNCTIONS

Like the values of single variables, it is also possible to pass the values of an array to function. When an array is used as an argument to a function, only the address of the array gets passed, not a copy of the entire array. In other words, when you pass any array to a function as argument then

it will pass as ***call-by-reference.*** So, if you make any change in the elements of function's array then the original array's elements will also be change.

Ex. W.A.P. to create a function named sort, which sorts the elements of the given array.

```
voi d sort(int n[ ], int sz)
{    for(int i=0; i<sz; i++)
     {   for (int j=i+1; j<sz; j++)
         {     if (n[i]>n[j])
               { n[i] = n[i] + n[j];
                 n[j] = n[i] - n[j];
                 n[i] = n[i] - n[j];
               }
}}
void main( )
{       clrscr( );
        int arr[50], i, size;
        printf("Enter size of the array :- ");
        scanf("%d",&size);
        for(i=0;i<size;i++)
        { printf("Enter any number :-");
          scanf("%d",&arr[i]);
        }
        sort(arr, size);
        printf("\n\nThe sorted array is :-");
        for (i=0 ; i<size ; i++)
                printf("\n%d",arr[i]);
        getch( );
}
```

FUNCTION OVERLOADING

'C' provides the facility to overload a function. A function decalared more than one time with same name but with different parameters is called function overloading.

For Example :-

```
int sum (int a, int b)                  // function 1
{    return a+b;  }
int sum (int a, int b, int c)           // function 2
{    return a+b+c; }
int sum (int a, int b, int c, int d)    // function 3
{    return a+b+c+d; }
void main( )
{
  printf("\n The sum of two number is %d",sum(6,7));  // calling of function 1
  printf("\n The sum of four number is %d",sum(8,7,3,7)); // calling of function 3
  printf("\n The sum of three number is %d",sum(9,6,9)); // calling of function 2
}
```

Chapter 8

Standard Library Functions

Like all other high level languages, 'C' provides function libraries containing **built in functions.** These library files are known as **header files** and these standard functions are extremely useful to the programmers**.**

A library is a collection of programs or functions which can be used by other programs.

'C' supports a rich library of standard functions. If you want to access a function, it is necessary to include its corresponding header file at the beginning of the program. Lets enjoy and know about some useful functions given below :

Functions of 'ctype.h'

(i) isalnum( ) :- This function checks that is given character is alphanumeric or not. An ***alphanumeric character*** is either a letter or a digit (**eg. 'A' to 'Z' or 'a' to 'z' or 0 to 9**). The special symbols are called non-alphanumeric characters (eg**. +, -, } etc.**). This function accepts a character or integer and returns either **zero** or **non-zero** value. The function returns non-zero value if the given character is alphanumeric otherwise it returns zero. For example :-

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main( )
{
        clrscr( );
        char ch;
        printf("Enter any character :- ");
        ch=getchar( );
        if (isalnum(ch))
                printf("\n\tThe character is alphanumeric");
        else
                printf("\n\tThe character is not alphanumeric");
        getch( );
}
```

(ii) isdigit( ) :- This function checks that is the given character is a digit or not. It also accepts a character as parameter and returns an integer. It returns a non-zero value if the given character is digit otherwise zero. For example :-

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main( )
{
        clrscr( );
```

```
        char ch;
        printf("Enter any character :- ");
        ch=getchar( );
        if (isdigit(ch))
                printf("\n\tThe character is digit");
        else
                printf("\n\tThe character is not digit");
        getch( );
}
```

(iii) isalpha ( ) :- This function checks that is the given character a letter. It returns a non-zero value if the character is a letter otherwise zero. The example of this function is :-

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main( )
{
        clrscr( );
        char ch;
        printf("Enter any character :- ");
        ch=getchar( );
        if (isalpha(ch))
                printf("\n\tThe character is a letter");
        else
                printf("\n\tThe character is not a letter");
        getch( );
}
```

(iv) islower ( ) :- It performs the test that is the given character a lower case letter or not. If the character is in lower case then it returns a non-zero value otherwise a zero value. The example of this function is :-

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main( )
{       clrscr(  );
        char ch;
        printf("Enter any character :- ");
        ch=getchar( );
        if (islower(ch))
                printf("\n\tThe character is lower case character");
        else
                printf("\n\tThe character is not lower case character");
        getch( );
}
```

(v) isupper ( ) :- It performs the test that is the given character an upper case letter or not. If the character is in upper case then it returns a non-zero value otherwise a zero value. The example of

this function is :-

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main( )
{       clrscr(  );
        char ch;
        printf("Enter any character :- ");
        ch=getchar( );
        if (isupper(ch))
                printf("\n\tThe character is upper case character");
        else
                printf("\n\tThe character is not upper case character");
        getch( );
}
```

(vi) tolower( ) :- It is a case conversion function. This function is used to change the upper case letter into lower case. All others are left unchanged. The example describes it in details :-

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main( )
{       clrscr(  );
        char ch;
        printf("Enter any character :- ");
        ch=getchar( );
        if(isalpha(ch))
            if (isupper(ch))
                printf("\n\tThe lower case letter is %c",tolower(ch));
            else
                printf("\n\tThe character is not an upper case letter");
        else
            printf("\n\tThe character is not alphabetic");
        getch( );
}
```

(vii) toupper ( ) :- It is also a case conversion function. This function is used to change the lower case letter into upper case. All others are left unchanged. For example :-

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main( )
{       clrscr(  );
        char ch;
        printf("Enter any character :- ");
        ch=getchar( );
        if(isalpha(ch))
```

```
            if (islower(ch))
                printf("\n\tThe upper case letter is %c",toupper(ch));
            else
                printf("\n\tThe character is not a lower case letter");
        else
            printf("\n\tThe character is not alphabetic");
        getch( );
}
```

Functions of 'string.h'

(i) strcpy ( ) :- This function copies one string to another. The syntax of this function is :

strcpy(destination, source);

It copies the source string to destination string. The copying action stops after the **NULL** character **'\0'** has been moved from source to destination. For example:-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main( )
{
    clrscr( );
    char d_string[20], s_string[20];
    printf("Enter the string :- ");
    gets(s_string);
    strcpy(d_string, s_string);
    printf("\n\n\t The given string is :- %s",s_string);
    printf("\n\n\t The copied string is :- %s",d_string);
    getch( );
}
```

(ii) strcat ( ) :- This is the string concatenate function. This function appends a source string to the end of destination string so that the length of the resulting string is equal to total of both string. For example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main( )
{
    clrscr( );
    char result[40]= ""; char
    blank[ ]= " ";
    charstr1[]= "Prashanta";
    char str2[]= "Computer";
    char str3[]= "Education";
    strcat(result, str1);
    strcat(result, blank);
    strcat(result, str2);
```

```
        strcat(result, blank);
        strcat(result, str3);
        printf("\n\n\t The resultant string is :- %s",result);
        getch( );
}
```

(iii) strlen ( ) :- This function is used to calculate the length of the given string. It takes a string as an argument and returns it's length as integer. For example :-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main( )
{
        clrscr( );
        char name[20];
        printf("Enter the string :- ");
        gets(name);
        printf("\n\n\t The length is :- %d",strlen(name));
        getch( );
}
```

(iv) strcmp ( ) :- This function is used to compare two strings for equality. The comparison begins with the first character of both strings and continues with subsequent characters until the end of the strings is reached or the compared characters differ from each other. This function accepts two strings and return an integer. It returns a value less than zero if string1 < string2, returns a value more than zero if string1 > string2 otherwise returns zero if string1 = string2. For example :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main( )
{
        clrscr( );
        char str1[20], str2[20];
        printf("Enter the first string :- ");
        gets(str1);
        printf("Enter the second string :- ");
        gets(str2);
        if (strcmp(str1, str2)==0)
            printf("\n\n\t The given strings are equal");
        else
            printf("\n\n\t The given strings are not equal");
        getch( );
}
```

(v) strlwr ( ) :- This function is used to converts the upper case string into lower case. When we use this function then it changes the original string. For example :-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main( )
{
        clrscr( );
        char str1[20];
        printf("Enter any string in upper case :- ");
        gets(str1);
        strlwr(str1);
        printf("\n\n\t The resultant string is %s",str1);
        getch( );
}
```

(v) strupr ( ) :- This function is used to converts the lower case string into upper case. It also changes the original string. For example :-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main( )
{
        clrscr( );
        char str1[20];
        printf("Enter any string in lower case :- ");
        gets(str1);
        strupr(str1);
        printf("\n\n\t The resultant string is %s",str1);
        getch( );
}
```

(v) strrev ( ) :- This function is used to reverse the given string. It also changes the original string. For example :-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main( )
{
        clrscr( );
        char str1[20];
        printf("Enter any string in upper case :- ");
        gets(str1);
        strrev(str1);
        printf("\n\n\t The resultant string is %s",str1);
        getch( );
}
```

Chapter 9

Structures

A Structure is a collection of one or more values, each potentially of the different data type. As we know, the elements in an array must be of the same data type, and we must refer to the entire array by its name. Each elements (called a member) in a structure can be of a different data type.

Defining a Structure :- The **struct** statement should be used to define a Structure. The keyword **struct** tells the compiler that a structure is being defined.

```
struct [struct tag]
{  member  definition  1;
       member definition2;
       member definitionn;
}[one or more structure variable];
```

For example, the following structure contains name, roll number and total marks of a student as components :-

```
struct student
{      char  name[20];
       int roll;
       int marks;
}data;
```

Referencing Structure Elements :- Once a structure variable has been defined, its members can be accessed through **'.'** (*the dot operator*). Or we can say "*The operator used to accesses structure variables is called the dot operator.*" For example, the following code assign a value in **roll** variable of **data** element :- ***data.roll=23;***

the syntax for accessing a structure elements is :-

structure-variable.element-name

The structure members are treated just like other variables. Therefore, to print the **roll** of **data**, we can write :-

```
printf("%d",data.roll);
```

In same manner to read **name** & **marks** we can write :-

```
gets(data.name);
scanf("%d",&data.marks);
```

However, the structure tag (e.g. **student** in the above given definition) can be omitted when only one structure variable is needed. That means that :-

```
struct
{      char name[20];
       int roll;
       int marks;
}data;
```

declares one variable data as defined by the structure preceding it, but *no more structure variable can be created with it.*

Ex.1) Declare the structure having the fields item name, item rate and item qty. Write a program to print the bill using structure variables.

Ans.)
```
struct item {
        char item_name[20];
        int item_rate,item_qty;
}a;
void main( )
{       clrscr( );
        printf("Enter Item Name      :-");
        gets(a.item_name);
        printf("Enter Item Rate      :-");
        scanf("%d",&a.item_rate);
        printf("Enter Item Qty       :-");
        scanf("%d",&a.item_qty);
        printf("\n\n Your Bill is Rs. %d /- Only",a.item_rate * a.item_qty);
        getch( ); }
```

Structure Variable Assignment :- You can assign a structure variable in two methods. One structure variable can be assigned at the time of declaring structure (within structure) or outside/separate from the structure. For example :

```
struct student                          struct student
{                                       { charname[20];
        char name[20];                      int roll,marks;
        int roll, marks;                };
}data;                                  student data;
```

Nested Structure:- C enables you to nest one structure definition within another. This technique saves time when you are writing programs that use similar structures. For example :

```
struct details {
        char date[10];
};
struct student
{       char name[20];
        int roll,marks;
        details dob;    };    // another structure
void main()
{       clrscr();
        student rec;
        printf("Enter Student's Name :-");
        gets(rec.name);
        printf("Enter Roll Number :-");
        scanf("%d",&rec.roll);
        printf("Enter Marks :-");
        scanf("%d",&rec.marks);
        printf("Enter Date of Birth :-");
        gets(rec.dob.date);
}
```

Structure with Arrays :- The array and structures can be combined together to form complex data objects. When you want to store the data of 10 students then you can define structure variable as an array of 10 elements as :-

```
struct student {
        char name[20];
        int roll, mark;
}data[10];
```

The reading and writing process is as same as an array like :-

```
for (int i=0;i<=9;i++)        {
        printf("Enter the name :-");
        gets(data[i].name);
        printf("Enter the roll number :-");
        scanf("%d",&data[i].roll);
        printf("Enter the mark :-");
        scanf("%d",&data[i].mark);
}
```

Passing Structure to Function :- The entire structure can be passed to the functions both ways *by value* and *by reference*. *Passing by value* is useful when the original values are not to be changed and *passing by reference* is useful when original values are to be changed.

Call-by-value :- When a structure is used as an argument to a function, the entire structure is passed using the standard call-by-value method. This means that any changes made to the contents of the structure inside the function to which it is passed do not affect the structure used as an argument.

```
#include<stdio.h>
#include<conio.h>
struct student
{
        char name[20];
        int roll, marks;
};
void display(student);
void main( )
{
        clrscr( );
        student data;
        printf("Enter students name :-");
        gets(data.name);
        printf("Enter roll number :-";
        scanf("%d",&data.roll);
        printf("Enter marks :-)";
        scanf("%d",&data.marks);
        display(data);
        getch();
}
```

```
void display(student var)
{
        printf("\nStudent Name :- %s",var.name);
        printf("\nRoll number :- %d",var.roll);
        printf("\nTotal Marks :- %d",var.marks);
}
```

Call-by-Reference :- When a structure is passed by reference the called function declares a reference for the passed structure and refers to the original structure elements through its reference. Thus, the called function works with the original values.

```
#include<stdio.h>
#include<conio.h>
struct num
{
        int first, second;
};
void swap(num &var);
void main()
{
        clrscr();
        num data;
        printf("Enter First Number :-");
        scanf("%d",&data.first);
        printf("Enter Second Number:-");
        scanf("%d",&data.second);
        swap(data);
        printf("\nValue of First after swap is %d",data.first);
        printf("\nValue of Second after swap is %d",data.second);
        getch();
}
void swap(num &var)
{
        var.first=var.first+var.second;
        var.second=var.first-var.second;
        var.first=var.first-var.second;
}
```

In the above program the function **swap()** creates reference **var** for structure variable **data** and thus, uses the original structure's value by names **var.** The changing in the values of **var** effects the original values of structure variable **data.**

Returning Structure from Function

Functions can also return structures like other types. The return type of the function is the same as that of the structure returned. For example, if the function ***calculation*** has to return a structure of type **employee**, it will be declared as:

```
struct employee
{
        int salary;
};
employee calculation(employee obj1, employee obj2)
{
        employee obj3;
        obj3.salary=obj1.salary + obj2.salary;
        return obj3;
}
void main( )
{
        clrscr( );
        employee total, emp1, emp2;
        printf("Enter salary of two employee :-");
        scanf("%d%d",&emp1.salary, &emp2.salary);
        total=calculation(emp1, emp2);
        printf("\n\n The total salary of both employees is %d",total.salary);
        getch( );
}
```

When the ***calculation( )*** function will call, the return value of the function is assigned to a variable which should be a structure variable of type **employee.**

Some Important Definitions

(i) Structure :- A structure is a collection of variables having different data types. We prefer array if we want to store same type of data instead of structure.
(ii) Structure tag :- A structure tag name can be used to declare variable of a structure type.
(iii) Structure member :- The member of a structure are called structure elements. Structure elements are referenced through the dot **'.'** or arrow **'->'** operators.
(iv) Arrow operator:- The arrow operator accessed structure member via a pointer to object.
(v) Nested Structure :- C++ enables you to nest one structure definition within another. This technique saves time when you are writing programs that use similar structures.
(vi) Copy Structure :- A structure can also be assigned to another structure only if both the structures are of same type.
(vii) **Structure and Function :-** When structure is passed to functions, structure variables are passed by copy as all other variables in C++. Therefore, if you pass a structure variable to a function that modifies one or more members of the structure, the calling function's structure variable is not automatically updated. This is method is also called *passing-by-value* or *call-by-value.*
(viii) Reference Structure :- Function can also return structures or its reference. Structure can be passed to function either by passing its elements individually or by passing the entire structure.

User Defined Data Types

(i) typedef :- You can define a new name of a specified data type. It provides an alternative name for the given data type. The syntax is :

typedef *type* new_name;

for example, you can create a new/alternative name for **float** type like :

typedef *float* total;

Now, you can also declare **float** type variables by giving it's alternative name before them. The above line tells the compiler to treat **total** as an alternative name for **float** like**:**

total a,b,c;

You can define an alias name of the given data type instead of replacing it. The new name is in addition to the existing type name. You still can create float variables using **float**.

***You can also declare structure variables through* typedef.** If you give **typedef** before **struct** keyword at the time of declaring a structure; the structure variable capable to declare more variables by giving it's name instead of structure tagname like :

```
typedef struct employee {
                            int a,b,c;
                        }var;
```

now you can declare more structure variable from **var** like :

var *value,value1,value2;*

#define Preprocessor Directive

#define preprocessor allows us to define symbolic names and constants like :

#define PI 3.14159 or #define NAME "Prashanta Computer Education" In the macro definition, it doesn't contain special characters or space and it cannot start with a number.

Macros :- You can also give an expression instead of a symbolic constant with **#define**, and this process is called macros. For example

```
#define SQR(x) x * x
#define CIRCLE (x) PI * x * x
```

A macro without arguments is treated like a symbolic constant. A macro with arguments has its arguments substituted for replacement text, when the macro is expanded.

Chapter 10

Data File handling

When you work on computer, it stores the program/information in files because files help in storing information permanently. You can access the matter of these files any time as you want. So, you can say, ***'A file itself is a bunch of bytes stored on the storage device (Hard-Disk, Floppy Disk etc.).'*** You can also stores the input data given by the user at program's runtime in the file permanently for work on it later. These files are called ***data files.***

To work on these files, the following library functions will be use:

fopen( )	Create or open a file for reading or writing.
fclose( )	Close a file after reading or writing it.
fseek( )	Seek to a certain location in a file.
ftell( )	Returns the byte number of the pointer position.
rewind( )	Rewind a file back to its beginning and leave it open.
rename( )	Rename a file.
remove( )	Delete a file.

C provides us a pre-defined structure named **FILE** to work on a file. All the above library functions works with this **FILE** pointer variable. So, first of all we have to specify a pointer variable of this structure which will reference to a data file. The pointer variable of this structure will create as :

FILE *fp;

You can also creates more than one variables of this structure by placing comma (,) between the variables. The **FILE** structure has been defined in the header file **<stdio.h>** so, It is necessary to include this file.

OPENING A FILE

For reading or writing the information in data file; first, we have to open that file by calling the function **fopen( )**. The **fopen( )** function returns a file pointer after opening the data file. For opening a data file the syntax is :

fp = fopen("<*filename*>", "<*mode*>");

When you opens any file, it is necessary to specify the mode of the file. The mode specifies

w	Always creates a fresh file. If the data file already exist it will overwrite the file otherwise create new.
a	Searches file for adding new records at the end of the file. It sets the pointer at the last of the file. If the file doesn't exist it creates new and write in it.
r	Open a file only for reading the records.
r+	Open a file for reading and writing contents of the existing file.
w+	Open and overwrite (or create) for reading and writing.
a+	Open a file for reading and appending.

If any of the mode unable to open the data file then all will returns **NULL.**
You can open a data file under these given modes according to your need. For example, if you want to open a file under writing mode then you have to give the following command :

```
fp = fopen ("DATA.DAT", "w");
fclose(fp);
fp=fopen ("TEMP.DAT", "r");
```

The above code let you to open two files in succession. This is necessary to close the first file before opening the second one, because a **FILE** structure pointer can be connected to only one file at a time.

CLOSING A FILE

After working on a file you have to close it for working further in the file pointer variable. The **fclose ( )** function will use to close a data file. The syntax is :

```
fclose (fp);
```

FILE RANDOM ACCESS

The **fseek( )** function provides the facility to move the pointer from one record to another. You can put the pointer at the starting, middle or last position of the file. The syntax of this function is:

fseek(<*file pointer*>, <*no of bytes*>, <*position*>);

There are three options under this function. In the first part, you have to specify a file pointer which is the reference of the file you have opened under it. The second part tell the compiler that how many bytes should the pointer be moved from a particular position. The third part of the function is used to move the pointer at a particular position.
Their are three position references used with **fseek( )** function are **SEEK_SET ('0'), SEEK_CUR ('1')** and **SEEK_END('2').** The **SEEK_SET** or **'0' (zero)** is used to move the pointer to the beginning of the file. The **SEEK_CUR** or **'1'** is used to move the pointer with reference of its current position and the **SEEK_END** or **'2'** is used to move the pointer to the end of the file.

THE ftell( ) FUNCTION

After placing the pointer to a particular position by **fseek( )** function you can find the byte number of the current position through **ftell( )** function. The **ftell( )** function returns a long int position number of the given file pointer variable. The syntax of this function is :

```
position = ftell ( fp );
```

READING OR WRITING OPERATION

After opening a file you have to write or read the data from the file. The **fread( )** function is used to read the data from the disk and placed it to the structure variable. The **fwrite( )** function will write the data to the disk which is containing by the structure variable. The syntax for both the function is same like as :-

fread / fwrite(&<*structure variable*>, <*size of one record*>, <*position*>, <*file pointer*>);

Ex.1) Write a program of Data Entry which enters the records to the file and read from it.

```
#include<stdio.h>
#include<conio.h>
struct employee
{
  char n[20];
  int sal;
};
void main()
{
   clrscr();
  int rec, i;
  char ch;
  FILE *fp;              // specifies the file pointer
  employee var;
  fp = fopen("DATA.TXT","w"); // opens the file in write mode
   do
   {
         printf("\nEnter any name :- ");
         fflush(stdin);
         gets(var.n);
         printf("Enter the salary :- ");
         scanf("%d",&var.sal);
         fwrite(&var,sizeof(employee),1,fp); // writes the record in file
         printf("\n\nDo you want to add more records (y/n):-");
         ch=getche();
  }while(ch!='n');
   fclose(fp);              // close the current open file
  fp=fopen("DATA.TXT","r"); // opens the file into read mode
  fseek(fp,0,2);           // moves pointer at last of file
   rec = ftell(fp) / sizeof(employee); // calculate total record numbers
   fseek(fp,0,0);          // moves pointer at begining of file

  // Display the records from the file

   for(i=1; i<=rec; i++)
   {
         fread(&var, sizeof(employee), 1, fp);
         printf("\n\nName is :- %s ",var.n);
         printf("\n Salary is :- %d",var.sal);
         getch();
   }
   fclose(fp);
}
```

MODIFYING RECORDS FROM THE FILE

If you want to modify any record from the data file then you have to open two files at a time. Read the data from the data file and compare it with the data you want to modify. If the data matches then input new records and put them into the second temporary file otherwise put same data into temporary file. After completing this task, close the both the files and remove the main data file from the disk with the help of **remove( )** function. After then change the name of temporary file with previous main data file with the help of **rename( )** function as shown below program :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct employee
{
  char n[20];
  int sal;
};
void main()
{
   clrscr();
   int rec, i;
  char ch, name[20];
  FILE *fp, *fp1;                    // specifies the two file pointers
  employee var;
  fp = fopen("DATA.TXT","a");  // opens the file in append mode
   do
   {
     printf("\nEnter any name :- ");
     fflush(stdin);
     gets(var.n);
     printf("Enter the salary :- ");
     scanf("%d",&var.sal);
     fwrite(&var,sizeof(employee),1,fp); // writes the record in file
     printf("\n\nDo you want to add more records (y/n):-");
     ch=getche();
  }while(ch!='n');
   fclose(fp);               // close the current open file
  fp=fopen("DATA.TXT","r"); // opens the file into read mode
  fseek(fp,0,2);            // moves pointer at last of file
   rec = ftell(fp) / sizeof(employee); // calculate total record numbers
   fseek(fp,0,0);           // moves pointer at starting of file
              // Display the records from the file
   for(i=1; i<=rec; i++)
   {
    fread(&var, sizeof(employee), 1, fp);
    printf("\n\nName is :- %s ",var.n);
```

```
    printf("\n Salary is :- %d",var.sal);
    getch();
   }
  rewind(fp);        // moves the pointer of DATA.TXT to the begining
  fp1 = fopen("Temp.TXT","w");     // opens the temporary file
  printf("Enter the name to modify :- ");
  fflush(stdin);
  gets(name);
   for (i=1; i<=rec; i++)
   {
    fread(&var, sizeof(employee), 1, fp);    // reads data from file
    if (strcmp(var.n, name)==0)         // compares the data
    {
      printf("\n\nEnter new name :- ");
       fflush(stdin);
      gets(var.n);
      printf("Enter the salary :- ");
      scanf("%d",&var.sal);
       fwrite(&var, sizeof(employee), 1, fp1); // writes new data to temporary file
    }
    else
       fwrite(&var, sizeof(employee), 1, fp1); // write same data to temporary file
   }
   fclose(fp);
  fclose(fp1);
  remove("DATA.TXT");
  rename("Temp.TXT", "DATA.TXT");
  fp = fopen("DATA.TXT","r");
   for (i=1; i<=rec; i++)
   {
    fread(&var, sizeof(employee), 1, fp);
    printf("\n\nName is :- %s ",var.n);
    printf("\n Salary is :- %d",var.sal);
    getch();
   }
   fclose(fp);
}
```

DELETING RECORDS FROM THE FILE

The deletion method is as same as modification but there is a little change in it. In this method you will read the data from the main data file and put it into the second temporary data file if the data don't match with the data which you want to delete. So, you have all the records except the those you want to delete in the temporary file. Now remove the original file and rename the temporary file with the original one. For code for deletion is as :-

```
.....
.....
.....
printf("Enter the name you want to delete :- ");
fflush(stdin);
gets(name);
 for (i=1; i<=rec; i++)
 {
  fread(&var, sizeof(employee), 1, fp);
  if (strcmp(var.n, name)!=0)
  {
    fwrite(&var, sizeof(employee), 1, fp1);
  }
 }
 fclose(fp);
fclose(fp1);
remove("DATA.TXT");
rename("Temp.TXT", "DATA.TXT");
fp = fopen("DATA.TXT","r");
 fseek(fp,0,2);            // moves pointer at last of file
 rec = ftell(fp) / sizeof(employee); // calculate total record numbers
 fseek(fp,0,0);           // moves pointer at starting of file
 for (i=1; i<=rec; i++)
 {
  fread(&var, sizeof(employee), 1, fp);
  printf("\n\nName is :- %s ",var.n);
  printf("\n Salary is :- %d",var.sal);
  getch();
 }
 fclose(fp);
}
```